RStudio Package Manager: Admin Guide

Version 0.3.0-4

Abstract

This guide will help an administrator install and configure RStudio Package Manager on a managed server. You will learn how to install the product on different operating systems, configure authentication, and monitor system resources.

Contents

1 Introduction

RStudio Package Manager organizes and centralizes R packages across your team, department, or entire organization.

Traditionally, R packages entered the organization from a variety of sources including CRAN, Bioconductor, Github, and even internally developed package sources. RStudio Package Manager empowers R users to access packages and reproduce environments while giving IT control and visibility into package use.

2 System Requirements

- Red Hat Enterprise Linux/CentOS Linux 6.0+
- Red Hat Enterprise Linux/CentOS Linux 7.0+
- Ubuntu 14.04
- Ubuntu 16.04

Package Manager should run on a server with a minimum of 2GB of RAM. Package sources can be lazily cached from CRAN or eagerly downloaded. Package Manager recommends 50-200GB of disk storage.

3 Where to install RStudio Package Manager?

RStudio Package Manager acts as a bridge between offline servers running R and upstream package sources like CRAN. RStudio Package Manager should be installed on a server in the network with outbound access to:

`https://rstudio-pm-sync.s3.amazonaws.com`

Offline servers running R will communicate with RStudio Package Manager across the internal network.

See Repository Syncing for more details.

4 Root Requirements

RStudio Package Manager does not run as root, see 5.3. Root is required to:

- Install RStudio Package Manager
- Start and Stop Package Manager via the service daemons
- Activate the RStudio Package Manager license

5 Getting Started

This chapter helps you install RStudio Package Manager on Ubuntu or Red Hat Enterprise Linux/CentOS Linux, learn to manage the server, and perform some initial configuration.

We built this checklist to guide you through that process.

1. Download RStudio Package Manager Installer
2. Install RStudio Package Manager - Ubuntu 5.1.1, Red Hat /CentOS 5.1.2
3. Initial Configuration - 5.2
4. Activate RStudio Package Manager License - D.4
5. Start RStudio Package Manager - 6.1

6. Add Repositories and Packages 9

5.1 Installation

5.1.1 Install Package Manager - Ubuntu

You will use `gdebi` to install Package Manager and its dependencies. It is installed via the `gdebi-core` package.

```
sudo apt-get install gdebi-core
```

You should have been provided with a `.deb` installer for RStudio Package Manager. If you only have a link to this file, you can use `wget` to download the file to the current directory.

```
wget https://download-url/rstudio-pm_0.3.0-4.deb
```

Once the `.deb` file is available locally, run the following command to install RStudio Package Manager.

```
sudo gdebi rstudio-pm_0.3.0-4.deb
```

This will install Package Manager into `/opt/rstudio-pm`

5.1.2 Install Package Manager - RedHat

You should have been provided with an RPM file which contains Package Manager. You can install this rpm file using `yum`. If you have only a link to the RPM file, you can use `wget` to download the file to the current directory.

```
sudo yum install --nogpgcheck rstudio-pm-0.3.0-4.x86_64.rpm
```

This will install Package Manager into `/opt/rstudio-pm/`.

5.2 Initial Configuration

Before RStudio Package Manager can run, you will need to update the configuration file located at `/etc/rstudio-pm/rstudio-pm.gcfg`. Complete the `Address` property within the `Server` section by specifying the URL used to access RStudio Package Manager by clients. For example:

```
Address = http://r-packages.example.com
```

The other configuration properties may also be set, see: A for details.

After updating the configuration file, follow the steps in D.4 to activate the Package Manager license. Once done, you're ready to restart the server 6.1 and add a repository with packages 9.

5.3 RunAs User

RStudio Package Manager runs under an unprivileged account.

The installer creates a user account and group named `rstudio-pm` and runs the RStudio Package Manager service under this account. If you wish to change the account under which the service runs, please see C.

6 Server Management

This section describes common administrative tasks for RStudio Package Manager.

6.1 Stopping and Starting

Occasionally it is necessary to start and stop the RStudio Package Manager service. Stopping and starting is handled by `systemd` or Upstart.

The specific stop/start commands depend on the service dameon. Commands for `systemd` and Upstart are listed below.

After a restart, any scheduled syncs that were missed during downtime will automatically begin. See 10.3 for details.

6.1.1 systemd (Red Hat/CentOS 7, Ubuntu 16.04)

`systemd` is a management and configuration platform for Linux. The newest versions of most major Linux distributions have adopted `systemd` as their default init system.

The RStudio Package Manager installer installs a `systemd` service called `rstudio-pm`, which causes the Package Manager to be started and stopped automatically when the machine boots up and shuts down. The `rstudio-pm` service is also automatically launched during installation.

Use the following commands to manually start and stop the server:

```
sudo systemctl start rstudio-pm
```

```
sudo systemctl stop rstudio-pm
```

You can restart the server with:

```
sudo systemctl restart rstudio-pm
```

If you wish to keep the server running without interruption, but reload the configuration, you can use the systemctl command to send a `HUP` signal:

```
sudo systemctl kill -s HUP --kill-who=main rstudio-pm
```

The `HUP` signal causes the server to re-initialize but does not interrupt the current processes or any of the open connections to the server.

> Use a `HUP` signal when your configuration changes are limited to properties marked as reloadable. See Appendix A to learn which settings may be reloaded via `HUP`. Perform a full restart of RStudio Package Manager when changing other properties.

You can check the status of the `rstudio-pm` service using:

```
sudo systemctl status rstudio-pm
```

And finally, you can use the `enable`/`disable` commands to control whether Package Manager should be run automatically at boot time:

```
sudo systemctl enable rstudio-pm
```

```
sudo systemctl disable rstudio-pm
```

6.1.2 Upstart (Ubuntu 14.04, Red Hat 6)

Upstart is a system used to automatically start, stop and manage services. The installer writes an Upstart configuration file to `/etc/init/rstudio-pm.conf`. This instructs the Upstart to initialize RStudio Package Manager as soon as the network is activated on the machine and stop when the machine is being shut down.

The Upstart configuration also ensures that the `rstudio-pm` process is respawned if the process unexpectedly terminates. However, in the event that there is an issue which consistently prevents RStudio Package Manager from being able to start (such as a bad configuration file), Upstart will give up on restarting the service after approximately 5 failed attempts within a few seconds. For this reason, you may see multiple repetitions of a bad RStudio Package Manager startup attempt before it transitions to the "stopped" state.

To start or stop the server, run the following commands, respectively.

```
sudo start rstudio-pm
```

```
sudo stop rstudio-pm
```

To restart the server you can run:

```
sudo stop rstudio-pm
sudo start rstudio-pm
```

The `restart` command re-initializes the server.

> We recommend `stop` and `start` over `restart` because some configuration changes are not incorporated into a restart. In particular, `restart` *does not* `re-read the Upstart definition at`/etc/init/rstudio-pm.conf`. Changes to this file need a`stop`and`start‘ to take effect.

If you wish to reload the configuration and keep the server and all R processes running without interruption, you can use the `reload` command:

```
sudo reload rstudio-pm
```

This command causes the server to re-initialize but does not interrupt the current processes or any of the open connections to the server.

> Use a `HUP` signal when your configuration changes are limited to properties marked as reloadable. See Appendix A to learn which settings may be reloaded via `HUP`. Perform a full restart of RStudio Package Manager when changing other properties.

To check the status or retrieve the process ID associated with `rstudio-pm`, run the following:

```
sudo status rstudio-pm
```

6.2 Upgrading

Upgrading RStudio Package Manager requires limited downtime. During an upgrade users will not be able to install packages. We recommend upgrading during a period of downtime.

The latest version is available on the download page along with release notes. The current version is available by running:

```
cat /opt/rstudio-pm/VERSION
```

To upgrade:

1. Download the latest `.rpm` or `.deb` file
2. Run the install command:

   Ubuntu:

`gdebi <rstudio-pm-version.deb>`

Red Hat/CentOS:

`yum install --nogpgcheck <rstudio-pm-version.rpm>`

The new version of RStudio Package Manager will install on top of an earlier installation. Existing configuration settings are respected. During installation the RStudio Package Manager service is restarted.

6.3 Purging RStudio Package Manager

You can fully remove RStudio Package Manager and all its data from your server using the following steps:

1. Stop the RStudio Package Manager service. (See 6.1 for details)
2. Uninstall the RStudio Package Manager package from your system.

   Ubuntu:

   `apt-get purge rstudio-pm`

   Red Hat/CentOS:

   `yum remove rstudio-pm`
3. Remove `/opt/rstudio-pm` if it still exists.
4. Remove logs from `/var/log/rstudio-pm*`
5. Purge the databases
   - When using SQLite, remove the `SQLite.Dir` directory. This has a default location of `/var/lib/rstudio-pm/db`.
   - When using PostgreSQL, drop the databases used by Package Manager. You may also wish to remove the PostgreSQL user associated with Package Manager.
6. Remove the `Server.DataDir` directory. By default, this is `/var/lib/rstudio-pm`.
7. Remove configuration files from `/etc/rstudio-pm` if they still exist.

6.4 Backups

We recommend including the RStudio Package Manager configuration file in `/etc/rstudio-pm` as well as the variable data directory which defaults to `/var/lib/rstudio-pm` in your system backups. If you have configured the databases to be stored outside the data directory, ensure that it is also included in the backup.

A running RStudio Package Manager server may be writing into the data directory. You should stop the RStudio Package Manager server before taking a backup.

```
sudo stop rstudio-pm
# Run appropriate backup steps here.
sudo start rstudio-pm
```

Your platform may need alternate commands to restart RStudio Package Manager. Please see Section 6.1 for instructions specific to your operating system version.

6.5 Privileged Ports

RStudio Package Manager listens on HTTP port 4242 by default. When you modify the `HTTP.Listen` or `HTTPS.Listen` configuration properties to use a privileged port under 1024, the service will fail to start, and you will see an error like the following in '`/var/log/rstudio-pm.log`:

```
2017/11/28 13:41:59 Error: Could not initialize the HTTP listener: listen tcp :80: bind: permission deni
```

If you wish to listen with HTTP or HTTPS on a privileged port (< 1024), you can grant the RStudio Package Manager binary permission to do so by issuing the following command as root:

```
sudo setcap 'cap_net_bind_service=+ep' /opt/rstudio-pm/bin/rstudio-pm
```

After issuing the above command, restart the service (see 6.1).

7 Licensing

When RStudio Package Manager is first installed on a system it operates in an evaluation mode for a period of time and then subsequently requires activation for continued use.

To determine the current license status of your system, you can use the following command:

```
/opt/rstudio-pm/bin/license-manager status
```

After purchasing a license to RStudio Package Manager, you will receive a product key that is used to activate the license on a given system.

You can activate your license key with the command:

```
/opt/rstudio-pm/bin/license-manager activate <product-key>
```

After activation, restart the RStudio Package Manager server.

```
stop rstudio-pm
start rstudio-pm
```

Your platform may need alternate commands to restart RStudio Package Manager. Please see Section 6.1 for instructions specific to your operating system version.

If you want to move your license of RStudio Package Manager to another system, you should first deactivate it on the old system.

```
/opt/rstudio-pm/bin/license-manager deactivate
```

7.1 Proxy Servers

If your server is behind an internet proxy, you may need to add an additional command line flag indicating the address and credentials required to communicate through the proxy. This may not be necessary if either the `http_proxy` or `all_proxy` environment variable is defined (these are read and used by the license manager when available).

If you do need to specify a proxy server explicitly you can do so using the `--proxy` command line parameter. For example:

```
/opt/rstudio-pm/bin/license-manager \
    --proxy=http://127.0.0.1/ activate <product-key>
```

Proxy settings can include a host-name, port, and username/password if necessary. The following are all valid proxy configurations:

```
http://127.0.0.1/
http://127.0.0.1:8080/
http://user:pass@127.0.0.1:8080/
```

If the port is not specified, the license manager will default to using port `1080`.

7.2 Offline Activation

If your system has no connection to the internet it's also possible to perform an offline activation. To do this, we recommend using our offline activation app which will walk you through the process: RStudio Offline Activation

You first generate an offline activation request as follows:

```
/opt/rstudio-pm/bin/license-manager activate-offline-request <product-key>
```

Executing this command will print an offline activation request to the terminal which you should copy and paste into an email to RStudio customer support (support@rstudio.com). You will receive a reply with a file attachment that can be used to activate offline as follows:

```
/opt/rstudio-pm/bin/license-manager activate-offline <activation-file>
```

After activation, restart the RStudio Package Manager server. Please see Section 6.1 for instructions specific to your operating system version.

If you want to renew your license of RStudio Package Manager or move it to another system you can also perform license deactivation offline. You can do this as follows:

```
/opt/rstudio-pm/bin/license-manager deactivate-offline
```

Executing this command will print an offline deactivation request to the terminal which you should send to support@rstudio.com.

You can also perform an offline check of your current license status using the following command:

```
/opt/rstudio-pm/bin/license-manager status-offline
```

7.3 Licensing errors

Package Manager uses the `license-manager` to determine if a valid license is available. Should an error occur when interacting with the license manager, Package Manager indicates that problem in the `/var/log/rstudio-pm.log` log. The license manager sends details about the error to the system messages (syslog). You should consult both locations should Package Manager be unable to obtain license status.

8 Files and Directories

8.1 Changing Ownership

Many of the files and directories mentioned in this chapter are, by default, owned by the `rstudio-pm` user. If you change the RunAs user for the RStudio Package Manager service, you will need to change ownership of these files and directories. See C for details on changing the RStudio Package Manager service RunAs user.

8.2 Program Files

The RStudio Package Manager installers place all program files into the `/opt/rstudio-pm` directory.

You should not need to change any files in the `/opt/rstudio-pm` hierarchy. Any alterations will be overwritten by subsequent re-installs or upgrades of RStudio Package Manager.

8.3 Configuration

The RStudio Package Manager configuration file is `/etc/rstudio-pm/rstudio-pm.gcfg`. This file is initially owned by `rstudio-pm` with permissions `0600`. You will edit this file to properly configure RStudio Package Manager for your organization.

A configuration management tool like Puppet or Chef can be used to maintain the `rstudio-pm.gcfg` file. We recommend that it remain owned by `rstudio-pm` and have permissions `0600`, as your configuration may need to contain passwords and other sensitive information.

RStudio Package Manager upgrades will not overwrite customizations to the `rstudio-pm.gcfg` file.

8.4 Server Log

The RStudio Package Manager server log is located at `/var/log/rstudio-pm.log`. This file is owned by `rstudio-pm` with permissions `0600`.

If `logrotate` is available when RStudio Package Manager is installed, a `logrotate` configuration will be installed. The default configuration is to rotate the logfile daily. The old log file will be stored alongside the original with a numeric extension, `.1`, `.2`, etc. The rotated log files are compressed after one day. The `.1` log file is retained uncompressed, but older logs are compressed. Most systems use `gzip` for compression, giving log files with extensions like `.2.gz`, `.3.gz`. Logs will be maintained for 30 days.

The manual for `logrotate` has more information.

8.5 Access Logs

The RStudio Package Manager HTTP access logs are located at `/var/log/rstudio-pm.access.log`. This file is owned by `rstudio-pm` with permissions `0600`. Log files are stored in Apache Combined Log Format. See http://httpd.apache.org/docs/2.2/logs.html#combined for a description of this format.

If `logrotate` is available when RStudio Package Manager is installed, a `logrotate` configuration will be installed. The default configuration is to rotate the logfile daily. The old logfile will be compressed and stored alongside the original log file with a `.1.gz` extension (then `.2.gz`, etc.). Logs will maintained for 30 days.

8.6 Variable Data

RStudio Package Manager manages R packages and repositories. All package source bundles are stored in the server's data directory. The Package Manager handles incoming requests for packages across repositories. Only a single copy of each package source is stored, even if the package is referenced in multiple repositories.

The RStudio Package Manager data directory also contains information used by the server to manage repositories including the RStudio Package Manager `SQLite` databases and encryption key if SQLite is used.

The default location for the RStudio Package Manager data directory is `/var/lib/rstudio-pm`. This can be customized by specifying an alternate `DataDir` in the `Server` section of your configuration file.

```
[Server]
DataDir = /mnt/rstudio-pm
```

The RStudio Package Manager SQLite databases **must** exist on local storage. If the location for `DataDir` is not local storage but a networked location over NFS, configure the `Dir` setting in the `SQLite` section of your server configuration file.

```
[Server]
DataDir = /mnt/rstudio-pm

[SQLite]
Dir = /var/lib/rstudio-pm/db
```

8.6.1 Permissions

`/var/lib/rstudio-pm` is owned by `rstudio-pm` with permissions `0701`.

9 Admin CLI

RStudio Package Manager is administered through a command-line interface (CLI). Administrators can use the CLI to create repositories and sources, add local packages to sources, and setup sync schedules for CRAN sources.

The CLI is installed at `/opt/rstudio-pm/bin/rspm`. The CLI uses the configuration defined in `/etc/rstudio-pm/rstudio-pm.gcfg` unless you specify an alternate configuration file with the `--config` flag.

9.1 Privileges

Users must be a member of the `rstudio-pm` group in order to use the RStudio Package Manager CLI. See C for instructions on changing the required group.

The RStudio Package Manager CLI uses a Unix domain socket for communicating with the RStudio Package Manager server. By default, the domain socket file is located at `/var/run/rstudio-pm/rstudio-pm.sock`. You can customize the location by configuring the `Server.SockFileDir` setting.

Any user invoking the RStudio Package Manager CLI must have read/write access to the Unix domain socket in order to communicate with the RStudio Package Manager server.

9.2 Location

The CLI is located at `/opt/rstudio-pm/bin/rspm`. We recommend that users make an alias, add this location to their PATH, or navigate to this directory if they will be running multiple commands in one session, for example:

```
alias rspm='/opt/rstudio-pm/bin/rspm'
```

9.3 Getting Help

Simply run `rspm` without any arguments to display the top-level help for the RStudio Package Manager CLI. For help with a specific command, you can use the `help` command. For example, to display help about the `create` command, you can use `rspm help create`.

9.4 Example

Here, we provide an example of configuring a new RStudio Package Manager to serve CRAN packages and local packages.

We will create a CRAN source and a local source. Then we will schedule and sync the CRAN source and add internal packages to the local source. Finally, we will create a repository that subscribes to both sources. For more information on sources and repositories, refer to B.2.

Throughout the example we will use the alias `rspm`, see 9.2.

9.4.1 Step 1: Create a local source

```
rspm create source --name=companyPkgs --type=local
```

Add the internal package `parcel_0.1.tar.gz`:

```
rspm add --source=companyPkgs --path=/path/to/parcel_0.1.tar.gz
```

9.4.2 Step 2: Create a CRAN source

```
rspm create source --name=companyCRAN --type=cran
```

Set the CRAN source to sync daily at 2am and fetch packages lazily:

```
rspm schedule --source=companyCRAN --cron="0 2 * * *" --sync-mode=lazy
```

We could wait until 2am for the first sync to occur, or we can sync now:

```
rspm sync --source=companyCRAN
```

9.4.3 Step 3: Create a repository

```
rspm create repo --name=companyRepo
```

Subscribe the repo to the two sources:

```
rspm subscribe --repo=companyRepo --source=companyPkgs
rspm subscribe --repo=companyRepo --source=companyCRAN
```

Verify that everything is correct:

```
rspm list sources
rspm list repos
rspm list sources --repo=companyRepo
rspm list packages --repo=companyRepo --search=parcel
```

Once setup, the repository will be available from the web UI and users will be able to install both CRAN and local packages.

10 Repositories and Sources

R repositories contain package source tar files and are the primary vehicle for organizing and distributing R packages. For more information on packages and repositories see B.

10.1 Repository Structure

R package repositories have a specific structure that enables client commands like `install.packages` to query the repository's contents and download packages.

A regular CRAN repository is just a set of files served from disk. RStudio Package Manager *does not create repositories on disk*. Instead, the Package Manager maintains a single copy of each package source and uses a database and specialized web server to handle HTTP requests from R.

Some example requests that can be served by the Package Manager:

PACKAGES file

`http://pkg-manager.example.com/repo/latest/src/contrib/PACKAGES`

This serves a PACKAGES file. The PACKAGES file for a repository is human-readable and contains information on each package available in the repository. Package Manager can also serve requests for PACKAGES.gz and PACKAGES.rds.

Package Source

`http://pkg-manager.example.com/repo/latest/src/contrib/package_2.1.0.tar.gz`

This request downloads the package source to the client.

Archived Package Source

`http://pkg-manager.example.com/repo/latest/src/contrib/archive/package/package_1.1.0.tar.gz`

This request downloads the tar file for an older, archived version of the package.

Most importantly, a Package Manager repository **is a CRAN-like repository** which means users can access and install packages using their regular R functions: `install.packages`, `available.packages`, `packrat`, and `devtools::install`.

10.2 Sources

10.2.1 About Sources

RStudio Package Manager repositories are composed of one or more **sources**. Sources are typed as either `local` or `cran`. Local sources can contain only local packages, while CRAN sources provide access to packages on CRAN.

10.2.2 Repositories with Multiple Sources

A repository can have more than one source. If you wish to serve both local packages and CRAN packages from a single repository, you can create a single repository that subscribes to both CRAN and local sources. For example:

- public (a repository)
  - internal (local source)
  - cran (CRAN source)

The "public" repository above gives users access to both local and CRAN sources, and its PACKAGES list could be accessed, for example, at http://pkg-manager.example.com/public/latest/src/contrib/PACKAGES. A repository subscribes to sources, which means that changes to a source will be reflected in the repository. For example, if an admin adds a new package to the `internal` source, users will automatically be able to access the new package via the `public` repository.

10.2.3 Package Conflicts Between Sources

If a repository has multiple sources and a package with the same name exists in both sources, RStudio Package Manager eliminates duplicates, giving preference to earlier sources. In the example repository above, if a package named "plumber" exists in both the "cran" and "internal" sources, the "plumber" package from the "internal" source would be served and listed since it is the first source for the repository. The same conflict resolution occurs as sources change. For example, in the sample above, if a new package is added to CRAN with the same name as an internal package, the internal package will continue to be served.

10.3 CRAN Sources

A primary use case for RStudio Package Manager is making packages in public repositories, like CRAN, available to enterprise users. To do so, an administrator can create a CRAN source and then synchronize the source with RStudio's curated version of CRAN.

10.3.1 What is RStudio's curated version of CRAN?

Package Manager doesn't download packages directly from CRAN. Instead, RStudio maintains a curated s3 bucket that contains metadata about CRAN and package tar files. The metadata is used to track CRAN's day-to-day changes.

During a sync, the metadata is downloaded to Package Manager. The metadata is compared against the Package Manager database to determine what changes need to be applied to the source. Package tarballs are then downloaded to the cache either eagerly or lazily depending on the sync mode.

Lazy sync mode is recommended.

10.3.2 Eager vs Lazy

Eager - If a CRAN source is setup for eager fetching, Package Manager will begin downloading packages as soon as the metadata is sync'd against the database. Downloading all of CRAN during an initial sync can take a significant amount of time, bandwidth, and disk space. However, downloading packages **does not block end users** from accessing repositories that subscribe to the source. End users can request any package as soon as the metadata is sync'd. If a user requests a package that is not already downloaded, the package will be immediately downloaded and served to the client. Packages are shared across Package Manager's sources. If a second CRAN source is created and sync'd, no additional packages will be downloaded. The benefit of eager fetching is that package are cached inside the local network.

Lazy - If a CRAN source is setup for lazy fetching, Package Manager downloads packages as the packages are requested by end users. Package Manager will still download the metadata from CRAN on the sync schedule to keep the Package Manager database updated. The database serves as the source of truth for package availability. The benefit of lazy fetching is a smaller footprint in terms of network bandwidth and disk space.

In either mode, each version of a package is only downloaded once. Package Manager always checks the local cache to see if the required tar file is already available.

Changing Modes

If you change a CRAN source from lazy to eager, the following occurs:

Lazy to Eager - Eager fetching is applied to future syncs. Packages added to the database during previous syncs are still fetched lazily. **Eager fetching is not applied to all packages**.

Eager to Lazy - Lazy fetching is applied to all future syncs. A sync with in-progress downloads will be completed.

10.3.3 How often should I sync my CRAN source?

The administrator who creates the CRAN source configures how often the CRAN source should sync and receive updates.

We recommend syncing every day.

CRAN adds or updates tens to hundreds of packages each day. Often R users will need the latest tools to perform their work.

Syncing a repository does not mean that new versions of packages are automatically deployed to end users. Syncing a repository does not run the risk of breaking functional code. **A repository specifies what packages are available, but the R user is in control of when and how to update the packages used by a project.**

Package Manager keeps track of old versions of packages as well. Old versions of packages are available in the repository's archive, and are listed in the Package Manager web UI. This allows users to roll back updates if necessary or install packages as they existed at a prior time.

11 Database

RStudio Package Manager supports multiple database options. Currently, the supported databases are SQLite and PostgreSQL.

Customize the `Database.Provider` property with a database scheme appropriate for your organization. See Section A.6 for details.

Here is a partial configuration which chooses to use SQLite.

```
[Database]
Provider = sqlite
```

11.1 SQLite

SQLite is the default database provider.

RStudio Package Manager will use SQLite database if the `Database.Provider` setting has a value of `sqlite` or if `Provider` is not present in the configuration file.

```
[Database]
Provider = sqlite
```

You can also specify the directory to store the SQLite file on your file system. This can be done by specifying `SQLite.Dir` in the configuration file.

```
[SQLite]
Dir = /mnt/rstudio-pm/sqlite
```

If this field is not specified, it will default to `{Server.DataDir}/db`. This location **must** exist on local storage.

If the location for `Server.DataDir` is not local storage but a networked location over NFS, configure the `SQLite.Dir` setting so it still resides on some local volume.

11.2 PostgreSQL

PostgreSQL is an available database provider which is more powerful and performant than SQLite.

You must provide your own Postgres server which will likely be a separate server from your RStudio Package Manager server (but not required). We currently support any 9.x version greater than or equal to 9.2. Your Postgres server does not have to be dedicated to RStudio Package Manager, but it must have its own dedicated database.

To use Postgres, select it as your provider with `Database.Provider = postgres`. You will also need to provide a fully qualified Postgres URL in `Postgres.URL`. The user credentials supplied in this URL must have read/write permissions to the database referenced at the end of url. Please ensure that you have already created a blank database with the name given at the end of your URL.

```
[Database]
Provider = postgres

[Postgres]
URL = "postgres://username:password@db.seed.co/rstudio-pm"
UsageDataURL = "postgres://username:password@db.seed.co/rstudio-pm-usage"
```

11.3 Usage Data

RStudio Package Manager relies on two databases by default. The primary database stores information needed to run the service including the arrangement of repositories, sources, and packages. Another database is used to record usage data like the number of times a package was downloaded.

If using SQLite, these two databases will be created automatically in the configured directory. If using PostgreSQL, you will need to define two different databases: `Postgres.URL` and `Postgres.UsageDataURL`.

If you do not wish to track usage data, you can disable this feature by setting `Server.UsageDataEnabled = false`. If disabled, usage data will not be tracked or displayed. You can use the `Server.UsageDataRetention` setting to alter the amount of usage data you wish to retain (the default is 365 days). Increasing this value will consume more disk space for the usage data database and may negatively impact performance slightly over time for busy servers.

12 Outbound Proxy

If you are syncing RStudio Package Manager to create your own CRAN repository (see 10.3), the server will need access to the internet to download manifest and package data. If you need to use an outbound proxy server, RStudio Package Manager automatically respects the `http_proxy` and `https_proxy` environment variables. To expose these environment variables to RStudio Package Manager, define them in `/etc/environment`.

13 Running with a Proxy

If you are running RStudio Package Manager behind a proxy server, you need to be sure to configure the proxy server so that it correctly handles all traffic to and from RStudio Package Manager. This section describes how to correctly configure a reverse proxy with Nginx or Apache HTTPD.

When RStudio Package Manager is behind a proxy, it is important to send the original request URL information to RStudio Package Manager so that it can generate fully qualified URLs and return them the requester. For this reason, when proxying to RStudio Package Manager, we recommend adding a header, `X-RSPM-Request`, to the request. This header value should be the absolute URL of the original request made by the user or browser (i.e. `https://rspm.company.com/some/path`)

Some proxies (like Amazon Web Services Elastic Load Balancer), do not make it possible to add custom headers. Because of this, if this header is not supplied, "best efforts" are made utilizing the standard headers `X-Forwarded-Proto`, `X-Forwarded-Host`, and `X-Forwarded-Port` to parse the original request URL. If your proxy removes a server prefix from the path, `X-Forwarded` headers will not work for your use case, and you should use `X-RSPM-Request`. If both `X-RSPM-Request` and `X-Forwarded` headers are supplied, `X-RSPM-Request` takes precedence.

13.1 Nginx Configuration

On Ubuntu, a version of Nginx that supports reverse-proxying can be installed using the following command:

```
apt-get install nginx
```

On Red Hat/CentOS, you can install Nginx using the following command:

```
yum install nginx
```

To enable an instance of Nginx running on the same server to act as a front-end proxy to RStudio Package Manager you would use a configuration like the following in your `nginx.conf` file. This configuration assumes RStudio Package Manager is running on the same host as Nginx and listening for HTTP requests on the `:4242` port. If you are proxying to RStudio Package Manager on a different machine or port, replace the `localhost:4242` references with the correct address of the server where RStudio Package Manager is hosted.

```
http {
    map $http_upgrade $connection_upgrade {
        default upgrade;
        ''      close;
    }
    server {
        listen 80;
        location / {
            proxy_set_header    X-RSPM-Request $scheme://$host:$server_port$request_uri;
            proxy_pass http://localhost:4242;
        }
    }
}
```

If you want to serve RStudio Package Manager from a custom path (e.g. `/rspm`) you would edit your `nginx.conf` file as shown below:

```
http {
    map $http_upgrade $connection_upgrade {
        default upgrade;
        ''      close;
    }
```

```
    server {
        listen 80;
        location /rspm/ {
            rewrite ^/rspm/(.*)$ /$1 break;
            proxy_set_header    X-RSPM-Request $scheme://$host:$server_port$request_uri;
            proxy_pass http://localhost:4242;
            proxy_redirect / /rspm/;
        }
    }
}
```

After adding these entries you'll then need to reload Nginx so that the proxy settings take effect.

On `systemd` systems (Red Hat/CentOS 7, Ubuntu 16.04):

```
systemctl restart nginx
```

On `upstart` systems (Ubuntu 14.04, Red Hat 6):

```
restart nginx
```

13.2 Apache Configuration

The Apache HTTPD server can act as a front-end proxy to RStudio Package Manager by first enabling three modules:

```
a2enmod rewrite
a2enmod headers
a2enmod proxy_http
```

The following configuration will permit proxying to RStudio Package Manager from the `:3737` port. Depending on the layout of your Apache installation, you may need the `Listen` and `VirtualHost` directives in different files.

```
Listen 3737

<VirtualHost *:3737>
    RewriteEngine on
    RequestHeader set X-RSPM-Request "%{REQUEST_SCHEME}s://%{HTTP_HOST}s%{REQUEST_URI}s"
    ProxyPass / http://172.17.0.1:4242/
    ProxyPassReverse / http://172.17.0.1:4242/
</VirtualHost>
```

You can serve RStudio Package Manager from a custom path (e.g. `/rspm`) with a configuration like the following:

```
Listen 3737

<VirtualHost *:3737>
    RewriteEngine on
    RedirectMatch ^/rspm$ /rspm/
    RequestHeader set X-RSPM-Request "%{REQUEST_SCHEME}s://%{HTTP_HOST}s%{REQUEST_URI}s"
    ProxyPass /rspm/ http://172.17.0.1:4242/
    ProxyPassReverse /rspm/ http://172.17.0.1:4242/
    Header edit Location ^/ /rspm/
</VirtualHost>
```

14 Security & Auditing

14.1 Browser Security

There are a variety of security settings that can be configured in RStudio Package Manager. Some of these settings are enabled by default but can be customized while others are opt-in. Below are some of the security features worth considering.

14.1.1 Guaranteeing HTTPS

If you can guarantee that your server should only ever be accessed over a TLS/SSL connection (HTTPS), then you can consider enabling the `Https.Permanent` setting. This elevates the security of your server by requiring that future interactions between your users and this server must be encrypted.

> Enabling this setting may keep users from being able to access your RStudio Package Manager instance if you later disable HTTPS or if your certificate expires. Use this setting only if you will permanently provide a valid TLS/SSL certificate on this server.

Behind the scenes, this makes two changes:

1. Introduces HTTP Strict Transport Security (HSTS) by adding a `Strict-Transport-Security` HTTP header with a `max-age` set to 30 days. HSTS ensures that your users' browsers will not trust a service hosted at this location unless it is protected with a trusted TLS/SSL certificate.
2. Enforces the `Secure` flag on cookies that are set. This prohibits your users' browsers from sending their RStudio Package Manager cookies to a server without an HTTPS-secured connection.

14.1.2 Content Sniffing

The `Server.ContentTypeSniffing` setting can be used to configure the `X-Content-Type-Options` HTTP header. This protects your users from a certain class of malicious uploads and is enabled by default.

When disabled (the default), the `X-Content-Type-Options` HTTP header will be set to a value of `nosniff` to tell browsers not to sniff the content type. If enabled, no such header will be provided.

14.1.3 Content Embedding

The `X-Frame-Options` HTTP header is used to control what content can be embedded inside other content in a web browser. The relevant attack is commonly referred to as a "clickjack attack" and involves having your users interact with a sensitive service without their knowledge.

> Some advertised values for this header are not supported across all browsers. RStudio Package Manager does not restrict the values of these headers.

14.1.4 Custom Headers

If you need to include additional HTTP headers that are not covered by any of the above features, you can include your own custom headers on all responses from RStudio Package Manager using the `Server.CustomHeader` setting.

This feature can be used to accommodate various other security practices that are not explicitly available as options elsewhere in Package Manager. For instance, X-XSS-Protection, Content Security Policy (CSP), HTTP Public Key Pinning (HPKP), and Cross-origin Resource Sharing (CORS) could all be configured using custom headers.

Custom headers are added to the HTTP response early during request processing. Values may later be overwritten or modified by other header settings. This includes both the security preferences described earlier in this chapter and other headers used internally by RStudio Package Manager. You should not depend on a custom header that conflicts with a header already in use by RStudio Package Manager.

The `Server.CustomHeader` takes a value of the header name and its value separated by a colon. Whitespace surrounding the header name and its value are trimmed. You can use this setting multiple times as in the following example:

```
[Server]
CustomHeader = "HeaderA: some value"
CustomHeader = "HeaderB: another value"
```

15 High Availability and Load Balancing

Multiple instances of RStudio Package Manager can share the same data in highly available (HA) and load-balanced configurations. In this document, we refer to these configurations as "HA" for brevity.

15.1 HA Checklist

Follow the checklist below to configure multiple RStudio Package Manager instances for HA:

1. Ensure that all node clocks are synchronized 15.2
2. Install and Configure the same version of RStudio Package Manager on each node - 5
3. HA requires using a PostgreSQL database. All nodes in the cluster must use the same PostgreSQL database.
4. Configure each server's `Server.DataDir` (8.6) to point to the same shared location. Be sure to read 15.3.3 for additional information on the recommended settings for the shared directory.

15.2 HA Time Synchronization Requirements

The clocks on all nodes in an HA configuration must be synchronized. We recommend configuring NTP for clock synchronization.

15.3 HA Limitations

15.3.1 Node Management

RStudio Package Manager nodes in an HA configuration are not self-aware of HA. The load-balancing responsibility is fully assumed by your load balancer, and the load balancer is responsible for directing requests to specific nodes and checking whether nodes are available to accept request

15.3.2 Database Requirements

RStudio Package Manager only supports HA when using a PostgreSQL database.

15.3.3 Shared Data Directory Requirements

RStudio Package Manager manages repository content within the server's data directory. This data directory must be a shared location, and each node's `Server.DataDir` must point to the same shared location. See 8.6 for more information on the server's data directory. We recommend and support NFS version 3 or 4 for file sharing.

RStudio Package Manager relies on being able to efficiently detect new files inside of the NFS-shared `DataDir`. By default, NFS clients are configured to cache responses for up to 60 seconds, which means that it can take up to a minute before an RStudio Package Manager service is able to respond to certain requests. For most deployments, this is an unacceptably long delay.

Therefore, we strongly recommend that you modify your NFS client settings for the mount on which you'll be hosting your `DataDir`. Typically, the best way to accomplish this is to set `lookupcache=pos` for your NFS mount, which will allow existing files to be cached but will contact the NFS server directly to check for the existence of new files. If this setting is not acceptable for your mount, you could alternatively consider shortening `acdirmax` or `actimeo` so that your client becomes aware of new files within, say, 5 seconds, instead of the default of 60.

15.4 Updating HA Nodes

When applying updates to the RStudio Package Manager nodes in your HA configuration, you should follow these steps to avoid errors due to an inconsistent database schema:

1. Stop all RStudio Package Manager nodes in your cluster.
2. Upgrade one RStudio Package Manager node. The first update will upgrade the database schema (if necessary) and start RStudio Package Manager on that instance - 6.2.
3. Upgrade the remaining nodes.

If you forget to stop any RStudio Package Manager nodes while upgrading another node, these nodes will be using a binary that expects an earlier schema version, and will be subject to unexpected and potentially serious errors. These nodes will detect an out-of-date database schema within 30 seconds and shut down automatically.

15.5 Downgrading

If you wish to move from an HA environment to a single-node environment, please follow these steps:

1. Stop all Package Manager services on all nodes
2. Reconfigure your network to route traffic directly to one of the nodes, unless you wish to continue using a load balancer.
3. If you wish to move all shared file data to the node, then
    1. Configure the server's `Server.DataDir` to point to a location on the node, and copy all the data from the NFS share to this location - 8.6
4. If you wish to move the databases to this node, install PostgreSQL on the node and copy the data. Moving the PostgreSQL databases from one server to another is beyond the scope of this guide. Please note that we do not support migrating from PostgreSQL to SQLite.
5. Start the Package Manager process 6.1

15.6 HA Details

15.6.1 Package Syncs

15.6.2 Adding Packages

15.6.3 Caching Content

A Configuration Options

This appendix documents the RStudio Package Manager configuration file format and enumerates the user-configurable options.

The RStudio Package Manager configuration file is located at `/etc/rstudio-pm/rstudio-pm.gcfg`. This configuration is read at startup and controls the operation of the service.

The RStudio Package Manager configuration file uses the **gcfg** (Go Config) format, which is derived from the Git Config format.

Here is an example of that format showing the different property types:

```
; Comment
[BooleanExamples]
property1 = true
property2 = off
property3 = 1

[IntegerExamples]
Property1 = 42
Property2 = -123

[DecimalExamples]
Property1 = 3.14
Property2 = 7.
Property3 = 2
Property4 = .217

[StringExamples]
Property1 = simple
Property2 = "quoted string"
Property3 = "escaped \"quote\" string"

[MultiStringExamples]
ListProperty = black
ListProperty = blue
ListProperty = green

[DurationExamples]
Property1 = 1000000000
Property2 = 500ms
Property3 = 1m15s  ; comment with a property
```

Comments always start with a semi-colon (`;`) and continue to the end of the line. Comments can be on lines by themselves or on a line with a property or section definition.

Configuration sections always begin with the name of the section bounded by square brackets. A section may appear multiple times and are additive with the last value for any property being retained. The following two configuration examples are equivalent.

```
[Example]
A = aligator
B = 2

[Example]
A = aardvark
C = shining
```

```
[Example]
A = aardvark
B = 2
C = shining
```

Each configuration property must be included in its appropriate section. Property and section names are interpreted case-insensitively.

Property definitions always have the form:

```
Name = value
```

The equals sign (`=`) is mandatory.

If a property happens to to be given more than once, only the last value is retained. The "multi" properties are an exception to this rule; multiple entries are aggregated into a list.

```
[MultiExample]
Color = black
Color = blue

[NonMulti]
Animal = cat
Animal = dog
```

If `Color` is a multi-string property, both the "black" and "blue" values are used. If `Animal` is a normal string property, only the value "dog" is retained.

Configuration properties all have one of the following types:

string A sequence of characters. The value is taken as all characters from the first non-whitespace character after equal sign to the last non-whitespace character before the end-of-line or start of a comment. Double-quotes (`"`) are supported, but usually unnecessary. A literal double-quote MUST be escaped and quoted itself like `QuotedValue = "J.R. \"Bob\" Dobbs"`.

multi-string A property that takes multiple string values. The property name is listed with each individual input value. For example, providing `Color = black` and `Color = blue` results in two separate values.

boolean A truth value. The values `true`, `yes`, `on`, and `1` are interpreted as true. The values `false`, `no`, `off`, and `0` are interpreted as false.

integer An integral value.

decimal A numeric value with an optional fractional component. Values with and without a decimal point are allowed.

duration A value specifying a length of time. When provided as a raw number, the value is interpreted as nanoseconds. Duration values can also be specified as a sequence of decimal numbers, each with optional fraction and unit suffix, such as `300ms`, `1.5h`, or `1m30s`.

Valid time units are `ns` (nanoseconds), `us` (microseconds), `ms` (milliseconds), `s` (seconds), `m` (minutes), and `h` (hours).

version A string representing a version. A version may have one to four numeric components, separated by periods or hyphens. Examples include `2`, `2.5`, `2.5.6`, `2.5.6.1`, and `2.5-6-11`.

bytesize A string representing a size in bytes. Valid examples include 10MB, 1 tb, 3 terabytes, and 2 GB.

Each configuration property documented in this appendix includes its description, data type, and default value.

Some properties are marked as "reloadable". Sending a `HUP` signal to the Package Manager process causes the on-disk configuration to be reread. The server is reconfigured with the latest values of these reloadable properties. See 6.1 for details about sending a `HUP` signal to your Package Manager process.

Use a `HUP` signal when your configuration changes are limited to properties marked as reloadable. Perform a full restart of RStudio Package Manager when changing other properties.

A.1 Server

The `Server` section contains configuration properties which apply across the whole of RStudio Package Manager and are not appropriate for the other sections, which are generally narrower.

The properties which follow all must appear after `[Server]` in the configuration file.

`DataDir` The directory where RStudio Package Manager will store its variable data.

Type: string

Default: `/var/lib/rstudio-pm`

`CacheDir` The directory containing the RStudio Package Manager cache.

Type: string

Default: `{Server.DataDir}/cache`

`DefaultFetchMode` The default fetch mode for CRAN synchronization. Must be either `eager` or `lazy`

Type: string

Default: `lazy`

`CRANTimeout` The timeout for downloading checkpoints or changes (JSON) from the CRAN manifest

Type: duration

Default: 30m

`FetchTimeout` The timeout for downloading a package tarball.

Type: duration

Default: 10m

`CacheTimeout` The amount of time we wait for files to appear in the cache directory after the database reports they are available. The default of 65 seconds should accommodate the NFS default directory caching limit of 60 seconds

Type: duration

Default: 65s

`Address` A public URL for this RStudio Package Manager server. Must be configured to enable features like including links to your content in emails.

Type: string

Default: *<empty-string>*

`ContentTypeSniffing` If disabled, sets the `X-Content-Type-Options` HTTP header to `nosniff`. When enabled, removes that header, allowing browsers to mime-sniff responses.

Type: boolean

Default: `false`

`ServerName` By default, Package Manager sets the `Server` HTTP header to something like `RStudio Package Manager v1.2.3`. This setting allows you to override that value.

Type: string

Default: *<empty-string>*

`AccessLog` Path to the file that RStudio Package Manager will use for its access logs. Disabled when empty.

Type: string

Default: `/var/log/rstudio-pm.access.log`

`CustomHeader` Custom HTTP header that should be added to responses from Package Manager in the format of `key: value`. The left side of the first colon in the string will become the header name; everything after the first colon will be the header value. Both will be trimmed of leading/trailing whitespace. This will always add a new header with the specified value; it will never override a header that Package Manager would otherwise have set. Multiple definitions can be used to provide multiple custom headers.

Type: multi-string

Default: *unspecified*

`FrameOptionsUI` The value for the `X-Frame-Options` HTTP header for the Package Manager UI and all other Package Manager pages. If empty, no header will be added.

Type: string

Default: `DENY`

`SockFileDir` RStudio Package Manager will use this directory to create a socket file for admin connections.

Type: string

Default: /var/run/rstudio-pm

`UsageDataEnabled` If true, will enable tracking of package downloads.

Type: boolean

Default: true

`UsageDataRetention` The number of days for which usage data will be retained. If `0`, data will be preserved forever.

Type: integer

Default: 365

A.2 Http

The `Http` section contains configuration properties which control the ability of RStudio Package Manager to listen for HTTP requests. RStudio Package Manager must be configured to listen for either HTTP or HTTPS requests (allowing both is acceptable).

These properties must appear after `[Http]` in the configuration file.

`Listen` RStudio Package Manager will listen on this network address for HTTP connections. The network address can be of the form `:80` or `192.168.0.1:80`. Either `Http.Listen` or `Https.Listen` is required.

Type: string

Default: *<empty-string>*

`NoWarning` Disables warnings about insecure (HTTP) connections.

Type: boolean

Default: `false`

A.3 Https

The `Https` section contains configuration properties which control the ability of RStudio Package Manager to listen for HTTPS requests. RStudio Package Manager must be configured to listen for either HTTP or HTTPS requests (allowing both is acceptable).

These properties must appear after `[Https]` in the configuration file.

`Listen` RStudio Package Manager will listen on this network address for HTTPS connections. The network address can be of the form `:443` or `192.168.0.1:443`. Either `Http.Listen` or `Https.Listen` is required.

Type: string

Default: *<empty-string>*

`Key` Path to a private key file corresponding to the certificate specified with `Https.Certificate`. Required when `Https.Certificate` is specified.

Type: string

Default: *<empty-string>*

`Certificate` Path to a TLS certificate file. If the certificate is signed by a certificate authority, the certificate file should be the concatenation of the server's certificate followed by the CA's certificate. Must be paired with `Https.Key`.

Type: string

Default: *<empty-string>*

`Permanent` Advertises to all visitors that this server should only ever be hosted securely via HTTPS. WARNING: if this is set to true – even temporarily – visitors may be permanently denied access to your server over an unsecured (non-HTTPS) protocol. This sets the `secure` flag on all session cookies and adds a `Strict-Transport-Security` HTTP header with a value of 30 days.

Type: boolean

Default: `false`

A.4 HttpRedirect

The `HttpRedirect` section contains configuration properties which control the ability of RStudio Package Manager to listen for HTTP requests and then redirect all traffic to some alternate location. This is useful when paired with an `Https.Listen` configuration.

These properties must appear after `[HttpRedirect]` in the configuration file.

`Listen` RStudio Package Manager will listen on this network address for HTTP connection and redirect to either the `HttpRedirect.Target` or `Server.Address` target location. The network address can be of the form `:8080` or `192.168.0.1:8080`. Useful when you wish all requests to be served over HTTPS and send users to that location should they accidentally visit via an HTTP URL. Must be paired with either `HttpRedirect.Target` or `Server.Address`.

Type: string

Default: *<empty-string>*

`Target` The target for redirects when users visit the `HttpRedirect.Listen` HTTP service. `Server.Address` is used as a redirect target if this property is not specified.

Type: string

Default: *<empty-string>*

A.5 Licensing

The `Licensing` section contains configuration properties which control how RStudio Package Manager interacts with its licensing system.

These properties must appear after `[Licensing]` in the configuration file.

`LicenseType` Enable remote or local validation. `local` is traditional activation, whereas `remote` uses floating licensing.

Type: string

Default: `local`

`RemoteRetryFrequency` When Package Manager loses its lease, it will begin automatically attempting to acquire a lease by `RemoteRetryFrequency`. Use a value of `0` to disable retries.

Type: duration

Default: `10`

A.6 Database

The `Database` section contains configuration properties which control the location of and how RStudio Package Manager interacts with its databases.

These properties must appear after `[Database]` in the configuration file.

`Provider` The type of database to use

Type: string

Default: sqlite

`MaxIdleConnections` The maximum number of database connections that should be retained after they become idle. If this value is less-than or equal-to zero, no idle connections are retained.

Type: integer

Default: `0`

`MaxOpenConnections` The maximum number of open connections to the database. If this value is less-than or equal-to zero, then there is no limit to the number of open connections.

Type: integer

Default: `0`

`ConnectionMaxLifetime` The maximum amount of time a connection to the database may be reused. If this value is less-than or equal-to zero, then connections are reused forever.

Type: duration

Default: `0`

A.7 SQLite

The `SQLite` section contains configuration properties which control the location of and how RStudio Package Manager interacts with the SQLite database.

These properties must appear after `[SQLite]` in the configuration file.

`Dir` The directory containing the RStudio Package Manager database. Must reference a directory on the local filesystem and not on a networked volume like NFS.

Type: string

Default: `{Server.DataDir}/db`

A.8 Postgres

The `Postgres` section contains configuration properties which control the location of and how RStudio Package Manager interacts with its postgres.

These properties must appear after `[Postgres]` in the configuration file.

`URL` The fully qualified URL to connect to a Postgres database

Type: string

Default: *<empty-string>*

`UsageDataURL` The fully qualified URL to connect to a Postgres database where usage information will be written.

Type: string

Default: *<empty-string>*

B Package Ecosystem

The R package ecosystem has a few key components.

B.1 Packages

Packages are the primary extension mechanism for R. They can be used to share functions, datasets, and documentation. An R package can exist in a few states:

B.1.1 Source

An R package is composed of a series of directories and files. The source of an R package is just a top-level directory containing the coponents of the package. Package authors work with source packages during development. Git(hub) repositories store source packages.

B.1.2 Bundle

A bundled package is a package thats been compressed into a single file. By convention, package bundles in R use the extension .tar.gz.

B.1.3 Binary

A binary package is the result of building a source package for a specific operating system. Binary packages are single files that are ready for installation on their specific operating systems.

B.1.4 Installed

An installed package is a binary package that has been decompressed into a package library and is ready for use by R.

B.2 Repositories

Repositories organize R packages for distribution to end users. Repositories contain package bundles and binaries that are organized in a specific way so that users can install packages from the repository using R's `install.packages` command. CRAN and Bioconductor are examples of R repositories.

B.3 Git(hub)

Many R package sources are stored in version controlled directories. A popular versioning tool is Git. Github, as an extension of Git, houses many package sources. The `devtools` R package includes convience functions for installing packages from the package source contained on a Git repository, including Github. Used in this manner, git repositories and Github are one way to distribute R packages, but Github and Git repositories *are not* R package repositories.

B.4 Libraries

End users of R typically interact with installed packages that live in libraries. Package libraries are just directories containing installed packages. When a package is requested by R, R searches the different library directories to find the installed package.

R libraries are very flexible. In the past, R users have set up libraries for specific projects or set up a system-wide library used across multiple projects. In multi-tentant servers it has been common to have both a system library shared by all users and user-specific libraries.

A best practice is to set up per-project libraries alongside a package cache.

C Changing RunAs User

C.1 Default Account

The installer creates a user account and group named `rstudio-pm` and runs the RStudio Package Manager service under this account. See 5.3 for more information.

C.2 Changing The RunAs Account (Service)

You can configure RStudio Package Manager to run under another account. The steps below serve as a guide for reconfiguring RStudio Package Manager to run under an account named `thor` with a primary group of `heroes` instead of the default `rstudio-pm:rstudio-pm`.

1. Stop the RStudio Package Manager service. See 6.1.

```
# Ubuntu 14 and Red Hat Enterprise Linux/CentOS Linux 6
sudo stop rstudio-pm

# Ubuntu 16 and Red Hat Enterprise Linux/CentOS Linux 7
sudo systemctl stop rstudio-pm
```

2. Create a new group and user account

```
sudo groupadd heroes
sudo useradd -r -g heroes -M -s /sbin/nologin thor
```

In order to use the CLI tool, a user must be a member of the primary group of the user that starts the RStudio Package Manager service.

In this example, RStudio Package Manager is started by the user `thor`. The primary group of the `thor` account is `heroes`, so users must be members of the `heroes` group to use the CLI.

3. Edit the service configuration

Ubuntu 14 and Red Hat Enterprise Linux/CentOS Linux 6

```
sudo vi /etc/init/rstudio-pm.override
```

Change these lines:

```
env RSTUDIO_PM_USER=thor
env RSTUDIO_PM_GROUP=heroes
```

Ubuntu 16 and Red Hat Enterprise Linux/CentOS Linux 7

```
sudo vi /etc/systemd/system/rstudio-pm.service.d/user.conf
```

Change these lines:

```
[Service]
User=thor
Group=heroes
```

4. Change ownership of files and directories

```
# Configuration file
sudo chown thor:heroes /etc/rstudio-pm/rstudio-pm.gcfg

# Log files
sudo chown thor:heroes /var/log/rstudio-pm.*

# Data directory (or `Server.DataDir`, if configured for a custom location)
sudo chown -R thor:heroes /var/lib/rstudio-pm

# Run directory
sudo chown -R thor:heroes /var/run/rstudio-pm

# If you have a custom `Sqlite.Dir` (e.g., `Sqlite.Dir = /database/directory`)
sudo chown -R thor:heroes /database/directory

# If you have a custom `Server.CacheDir` (e.g., `Server.CacheDir = /path/to/cache`)
sudo chown -R thor:heroes /path/to/cache
```

5. Remove remaining domain socket file (if any)

```
sudo rm /var/run/rstudio-pm/rstudio-pm.sock
```

6. Start the RStudio Package Manager service. See 6.1.

```
# Ubuntu 14 and Red Hat Enterprise Linux/CentOS Linux 6
sudo start rstudio-pm

# Ubuntu 16 and Red Hat Enterprise Linux/CentOS Linux 7
sudo systemctl daemon-reload # Reload the systemd process
sudo systemctl start rstudio-pm
```

7. Verify that the `rstudio-pm` service is running under the `thor` account.

```
ps -axj | grep `id -u thor`
```

8. Check `/var/log/rstudio-pm.log` to verify that the server started up with no errors.

C.3 Using the New RunAs Account (CLI)

After changing the service RunAs user, your CLI users must be members of the `heroes` group. For example:

```
sudo useradd -g heroes hulk
sudo passwd hulk
su hulk
/opt/rstudio-pm/bin/rspm <command>
```

D Manual Installation for Minimal Root Use

D.1 Discussion

The RStudio Package Manager installer requires root privileges and installs the RStudio Package Manager to `/opt/rstudio-pm`. Additionally, it configures services for upstart or systemd (depending on your OS), adds log files to `/var/log`, adds a configuration file under `/etc/rstudio-pm`, creates an `rstudio-pm` user account and group, and places a logentries configuration.

It is possible to extract the RStudio Package Manager files manually and configure the service to use configuration and data files at a custom location. Most of these steps can be performed without root privileges.

D.2 Extracting Files

The first step is to extract the files from the installer. Follow the instructions for your operating system, below.

D.2.1 Extracting Files (Ubuntu)

```
mkdir rspm
ar x rstudio-pm_<version>_amd64.deb
tar -xzvf data.tar.gz -C rspm/
cd rspm
```

D.2.2 Extracting Files (Red Hat Enterprise Linux/CentOS Linux)

```
mkdir rspm
cd rspm
rpm2cpio ../rstudio-pm-<version>.x86_64.rpm | cpio -div
```

D.3 Create Directories

Create the directories you need to run RStudio Package Manager.

```
mkdir log
touch log/rstudio-pm.log
touch log/rstudio-pm.access.log
mkdir run
mkdir data
```

D.4 Licensing

The license manager used by RStudio Package Manager supports userspace license activation. If you wish to activate a system-wide license, root privileges are required. See D.4 for more details on system-wide licensing.

```
# Initialize a trial in userspace
./opt/rstudio-pm/bin/license-manager initialize --userspace
```

```
# Activate a userspace license key, if you have a license key
./opt/rstudio-pm/bin/license-manager activate --userspace <key>
```

D.5 Edit config file

Next, edit the RStudio Package Manager configuration file to point to the directories and access log you created.

```
vi ./etc/rstudio-pm/rstudio-pm.gcfg
```

Add the following lines to the `[Server]` configuration section.

```
[Server]
DataDir = <path-to-unbundled-rspm-dir>/data
SockFileDir = <path-to-unbundled-rspm-dir>/run
AccessLog = <path-to-unbundled-rspm-dir>/log/rstudio-pm.access.log
```

D.6 Start the RStudio Package Manager Service

```
./opt/rstudio-pm/bin/rstudio-pm --config \
./etc/rstudio-pm/rstudio-pm.gcfg >> ./log/rstudio-pm.log 2>&1
```

D.7 Use the CLI to Manage RStudio Package Manager

```
./opt/rstudio-pm/bin/rspm --config ./etc/rstudio-pm/rstudio-pm.gcfg shell
```